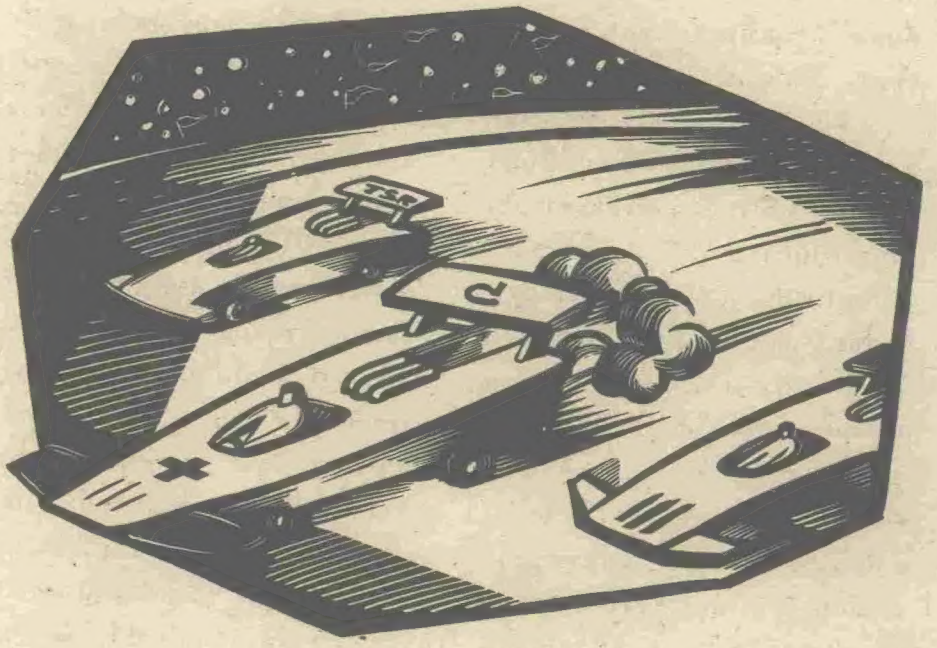
กิติพงษ์ ฤทธินุตร

เรียน/เขียน TSR กับ C ตอนที่ 2

ศึกษาการสร้าง TSR ที่ใช้ในการเขียนโปรแกรมจากแม่แบบที่สร้างขึ้นเอง



ที่ผ่านมาแล้วตอนหนึ่งนั้นเป็นเรื่องทั่ว ๆ ไปเกี่ยวกับโปรแกรมประเภท TSR (Terminate-and-stay-Resident) ไม่ว่าจะเป็นเรื่องของลักษณะโปรแกรม วิธีทำงานหรือโครงสร้างในการเขียนโปรแกรมรวมถึงเรื่องของอินเตอร์รัพต์บางชุดของดอสที่จำเป็นจะต้องใช้เพื่อตรวจสอบสถานะของดอสว่ามีความปลอดภัยพอที่จะให้ TSR ทำงาน (pop-up) หรือไม่ นอกจากนี้ก็ยังมีเรื่องเกี่ยวกับคอมพายเลอร์เทอร์โบซีซึ่งได้สร้างฟังก์ชันต่าง ๆ ไว้เพื่อสนับสนุนการเขียนโปรแกรมประเภทฝังตัวอยู่ในหน่วยความจำและให้สามารถเขียนคำสั่งได้สะดวกขึ้นโดยไม่ต้องใช้ภาษาแอสเซมบลีซึ่งทั้งหมดที่ได้พูดถึงกันไปแล้วนั้นคงจะช่วยสร้างให้มีความเข้าใจและทำให้รู้จักกับ TSR บ้างพอสมควร

ในช่วงต่อไปของบทความนี้จะเป็นเรื่องของการสร้าง TSR ที่จะใช้เป็นตัวอย่างในการเขียนโปรแกรมประเภทนี้ ซึ่งน่าจะเป็นประโยชน์ต่อการเขียน TSR ขึ้นใช้เองตามความต้องการสำหรับ TSR ที่จะสร้างขึ้นมาเป็นแม่แบบนี้ จะมีขีดความสามารถหลาย ๆ อย่างคือ

- จะมีระบบที่ใช้ในการแนะนำวิธีใช้คำสั่งดอสบางชุด (pop-up help system)
- มีส่วนที่จะช่วยแสดงรายละเอียดที่มีอยู่ในไฟล์ (file list utility)
- สามารถใช้ในการแสดงรายชื่อไดเรคทอรี (directory list utility)
- จะทำหน้าที่เปลี่ยนค่าตัวเลขระหว่างเลขฐาน 16 และ เลขฐาน 10 (number base converter)

ซึ่งขีดความสามารถเหล่านี้สามารถจะนำมาใช้ให้เป็นประโยชน์กับงานคอมพิวเตอร์ของคุณได้ และยังสามารถที่จะเปลี่ยนแปลง แก้ไข เพิ่มเติม โปรแกรมนี้ให้เหมาะกับความต้องการของคุณได้

สร้างโปรแกรม TSR

มาถึงตรงนี้เป็นอันว่าเราได้พูดถึงเนื้อหาที่เป็นหลักการเกี่ยวกับโปรแกรม (Application) ประเภท TSR มาพอสมควรแล้ว ซึ่งคิดว่าคงจะเพียงพอให้สามารถจะนำมาใช้เพื่อพัฒนาโปรแกรมของเราเองได้ โดยที่โปรแกรมตัวอย่างของเรานี้นอกจากจะเป็น TSR ที่สามารถนำไปใช้งานได้จริง ๆ แล้ว ยังสามารถใช้เป็นต้นแบบที่จะนำไปพัฒนาเพื่อสร้าง TSR ของเราเองได้ด้วย สำหรับ TSR ที่จะใช้เป็นต้นแบบนั้นจะมีชื่อว่า 'SCTSR' (Super Charging TSR)

รูปที่ 1 การกำหนดค่าเริ่มต้นให้กับฟังก์ชัน main ()

```
main()
{
  union REGS r;
  struct SREGS s;

  if(!load_help()) {  /* load the help file */
    /* You may want to take other action here. */
    exit(1);
  }

  /* see if already loaded */
  old_int64 = getvect(0x64); /* unused interrupt */
  if(!old_int64) setvect(0x64, 1); /* set a flag */
  else {
    printf("TSR already loaded");
    exit(0);
  }

  /* obtain the dos_active flag address */
  r.h.ah = 0x34;
  int86x(0x21, &r, &r, &s);
  dos_active = MK_FP(s.es, r.x.bx);

  /* get old ISR addresses */
  old_key = getvect(9); /* keystroke int */
  old_int28 = getvect(0x28);  /* int 28 */
  old_int8 = getvect(8);  /* timer interrupt */

  /* re-route interrupt vectors to our handlers */
  setvect(9, tsr_keystroke);
  setvect(0x28, dos_idle);
  setvect(8, new_int8);

  printf("SC-TSR installed\n");
  printf("F1: help    F2: file list    F3: directory");
  printf("    F4: base convertor\n");

  key = 0;
  set_vid_mem();  /* get address of video memory */

  /* initialize all windows */
  make_tsrwindow(0, " Help ('quit' to exit, 'topics' for topics) ",
              0, 0, 79, 24, BORDER);
  make_tsrwindow(1, " File Lister ", 0, 0, 79, 16, BORDER);
  make_tsrwindow(2, " Directory ", 0, 0, 79, 12, BORDER);
  make_tsrwindow(3, " Base Converter ", 0, 0, 50, 3, BORDER);

  keep(0, 4000); /* terminate and stay resident */
}
```

วิธีทำงาน

กำหนดค่าเริ่มต้นให้ SCTSR ส่วนที่ใช้ทำหน้าที่กำหนดค่าเริ่มต้น (initialization portion) ให้กับโปรแกรม TSR นั้น ส่วนใหญ่ก็จะทำไว้ในฟังก์ชัน 'main ( )' ของโปรแกรม ซึ่งค่าต่าง ๆ ที่ได้กำหนดเอาไว้ให้เป็นค่าเริ่มต้นของ TSR ได้แสดงไว้ในรูปที่ 1

จากชุดคำสั่งของโปรแกรมที่ได้แสดงไว้นั้นจะเห็นว่าสิ่งแรกที่ฟังก์ชัน 'main ( )' จะทำคือการเรียกใช้ฟังก์ชัน 'load_help ( )' ซึ่งฟังก์ชันนี้จะทำหน้าที่อ่าน help file (ที่เราจะได้สร้างขึ้นในช่วงต่อไป) และกำหนดค่าตัวชี้ (pointer) เพื่อให้ชี้ไปที่ข้อความที่ต้องการ (จะกล่าวถึงโดยละเอียดอีกครั้งในเรื่องของ pop-up help system)

สิ่งสำคัญที่จะต้องทำความเข้าใจในช่วงนี้คือ การเรียกใช้ (load) help file นั้น อาจจะมีเหตุการณ์ที่ทำให้ไม่สามารถเรียกใช้ได้สำเร็จก็ได้ (เช่นไฟล์ที่ต้องการนั้นอาจไม่ได้อยู่ในไดเรคทอรีที่กำลังใช้งานอยู่ขณะนั้น) เพราะฉะนั้นจะต้องจัดการกับเหตุการณ์เหล่านี้ก่อนที่จะกำหนดค่าเริ่มต้นต่าง ๆ ให้กับฟังก์ชันที่ใช้ ตัวอย่างเช่นไม่ควรจะคืนค่า (reset) ให้กับ interrupt vector จนกว่าจะได้จัดการกับงานทุกอย่างที่จะใช้ป้องกันความผิดพลาดหรือล้มเหลวของฟังก์ชัน

ที่จะเรียกใช้นั้นเรียบร้อยแล้ว เพราะการเรียกใช้ TSR นั้นอาจมีผลกระทบต่อการทำงานของโปรแกรมอื่นที่ทำอยู่ก่อนได้ (ทำให้งานเดิมเสียหาย) ฉะนั้นการจัดการทุกสิ่งทุกอย่างให้อยู่ในสภาพพร้อมและถูกต้อง จึงจำเป็นอย่างยิ่งสำหรับการทำงานของ TSR

เรื่องแรกเลยต้องจัดเตรียมเพื่อเป็นการป้องกันการเรียกโปรแกรม TSR ซ้ำอีก ซึ่งต้องทำการตรวจสอบดูว่าโปรแกรมที่ต้องการนั้นถูกเรียกเข้ามาเก็บไว้ในหน่วยความจำแล้วหรือยัง (เป็นชุดคำสั่งที่อยู่ถัดจากการเรียกใช้ help file) เพราะไม่ควรที่จะเรียกใช้ TSR มากกว่าหนึ่งครั้ง ถึงแม้ว่าในทางเทคนิคจะไม่ถือว่าเป็นเรื่องผิด แต่ก็ไม่ใช่วิธีการที่ดีที่จะเรียก TSR ซ้ำ เพราะการเรียกใช้แต่ละครั้งก็จะทำให้มีการใช้เนื้อที่ในหน่วยความจำเพิ่มขึ้นตามจำนวนที่ TSR ต้องการใช้งานและสิ่งที่จะช่วยตรวจสอบการฝังตัวของโปรแกรมได้คืออินเตอร์รัพต์ 0×64 ซึ่งว่างอยู่ (ไม่ถูกนำเอาไปใช้ในงานอื่น) โดยสามารถจะใช้หน่วยความจำในตารางอินเตอร์รัพต์ที่กันเอาไว้ให้กับอินเตอร์รัพต์ตัวนี้เพื่อใช้ทำหน้าที่เป็นแฟลก (flag) แสดงให้รู้ว่ามีการเรียกใช้โปรแกรม TSR เอาไว้แล้วหรือยัง

ก่อนที่ TSR จะเปลี่ยนทิศทางของอินเตอร์รัพต์ต่าง ๆ จะทำการตรวจสอบค่าของอินเตอร์รัพต์ 0×64 ที่ตำแหน่งเวกเตอร์ (address vector) และถ้าพบว่ามีค่าเป็น '0' ก็จะตั้งข้อสันนิษฐานว่าเป็นการเรียกใช้ TSR เป็นครั้งแรกและจะเปลี่ยนค่านี้ให้เป็น '1' อย่างไรก็ตามถ้าพบว่ามีค่าเป็น '1' อยู่แล้วจะตั้งข้อสมมติฐานว่ามีการเรียกใช้ TSR แล้ว และเลิกใช้ (exit) ไปแล้วก่อนที่จะเปลี่ยนทิศทางของอินเตอร์รัพต์ต่าง ๆ เพื่อทำงานต่อไป เพราะถ้าเป็นการออกจากโปรแกรมหลังจากที่ได้เปลี่ยนทิศทางของอินเตอร์รัพต์เวกเตอร์ แต่ก่อนการเรียกใช้ฟังก์ชัน '0×49' เพื่อฝังตัวในหน่วยความจำ จะทำให้ไม่รู้ว่าจะไปเริ่มต้นทำงานที่ไหนเมื่อมีการเรียกใช้อินเตอร์รัพต์อีก

การทำงานของฟังก์ชัน 'main ( )' ในช่วงต่อไป คือการจัดหาตำแหน่งให้กับดอส DOS แอดตีฟแฟลกด้วยการเรียกใช้ดอสฟังก์ชัน 0×34 และเมื่อค่าของรีจีสเตอร์ 'ES' และ 'BX' ถูกดึงเข้ามาใช้แล้วก็จะทำการสร้าง 'far pointer' โดยใช้ฟังก์ชัน 'MK_PF ( )' ของเทอร์โบซีและจะนำเอาพอยน์เตอร์ตัวนี้ไปเก็บไว้ในดอสแอดตีฟแฟลก

ในขั้นต่อไปจะเป็นการจัดเก็บตำแหน่งของ ISR สำหรับอินเตอร์รัพต์ 8, 9 และ 0×28 ที่มีอยู่เดิมโดยใช้ฟังก์ชัน geavact ( )' ของเทอร์โบซี ซึ่ง TSR ชุดใหม่จะใช้ตำแหน่งเหล่านี้เพื่อจะเรียกอินเตอร์รัพต์ชุดเดิม ฉะนั้นเมื่อเขียนโปรแกรม TSR ก็จะต้องแน่ใจว่าได้ทำการเชื่อมโยง TSR ที่สร้างขึ้นใหม่เพื่อใช้แทนของเดิมกับของเดิมที่เก็บเอาไว้ในตำแหน่งอื่นเพราะอินเตอร์รัพต์บางชุดเช่น คีย์บอร์ดอินเตอร์รัพต์และไทม์เมอร์อินเตอร์รัพต์นั้น ISR ชุดเดิมจะทำหน้าที่ให้บริการที่มีความสำคัญต้องการใช้ระบบมาก

นอกจากนี้ในกรณีที่มี TSR มากกว่า 1 โปรแกรมที่ฝังตัวอยู่ในหน่วยความจำก็จะต้องแน่ใจว่าได้ทำการเชื่อมโยงกันเอาไว้อย่างดี และเมื่อการใช้งาน TSR เสร็จแล้วก็จะต้องนำเอาตำแหน่งของ TSR ของเดิมไปเก็บไว้แทนชุดที่นำมาใช้แทนนั้นด้วย

เมื่อได้ทำการเก็บตำแหน่งเดิมของ TSR สำหรับอินเตอร์รัพต์ 8,9 และ 0×28 แล้วสามารถจะเก็บตำแหน่งของรูทีน (routines) ชุดใหม่เอาไว้แทนโดยใช้ฟังก์ชัน 'setvect ( )' โดยมีชื่อใหม่ของ 'keyboard function' เป็น 'trs_keystroke ( )' ส่วนชื่อใหม่ของ 'timer function' คือ 'new_ints ( )' และชื่อใหม่ของอินเตอร์รัพต์ 0×28 คือ 'dos_idle ( )' และเมื่อทำขั้นตอนเหล่านี้เสร็จแล้วก็จะทำให้ฟังก์ชันใหม่เหล่านี้พร้อมที่จะทำงานแทนฟังก์ชันชุดเดิม ฉะนั้นจะต้องแน่ใจว่าได้กำหนดค่าเริ่มต้น (initialization) ในการทำงานของรูทีนชุดใหม่เอาไว้อย่างถูกต้องเหมาะสม

ในคอมพายเลอร์เทอร์โบซีนั้นตัวพอยน์เตอร์ต่าง ๆ ที่จะชี้ไปที่ ISR ชุดเดิมจะถูกประกาศ (declare) เอาไว้ดังนี้

```
void interrupt (*old_int28) ( ) ;
void interrupt (*old_key) ( ) ;
void interrupt (*old_int8 ( )) ;
```

ซึ่งถ้าใช้คอมพายเลอร์ตัวอื่นก็อาจจะต้องเปลี่ยนแปลงวิธีการเขียนคำสั่งเหล่านี้ให้ถูกต้องตามกฎเกณฑ์ของคอมพายเลอร์นั้น

ส่วนงานขั้นต่อไปคือการกำหนดค่าเริ่มต้นของตัวแปร (global variable key) ซึ่งจะใช้เพื่อส่งผ่านรหัสแป้นพิมพ์ (key codes) ให้กับฟังก์ชัน 'dos__idle ( )' และ 'new__int8 ( )' หลังจากนั้นจะเรียกใช้ฟังก์ชัน 'set__vid__mem ( )' เพื่อกำหนดให้ global far pointer ชื่อ 'vid__mem' ชี้ไปที่จุดเริ่มต้นของหน่วยความจำแสดงผล (video memory) ที่ใช้สำหรับผลลัพธ์ที่เป็นข้อความ (text output) ซึ่งชุดคำสั่ง (โปรแกรม) ของฟังก์ชันนี้ได้แสดงเอาไว้ร่วมกับฟังก์ชัน 'video__mode ( )' ในรูปที่ 2

สำหรับ TSR ที่เป็นตัวอย่างของเรานี้จะมีป๊อปอัพฟังก์ชันอยู่ 4 ชุด ซึ่งในช่วงต่อไปเราจะได้พูดถึงชุดคำสั่งในฟังก์ชัน 'main ( )' ที่จะทำหน้าที่กำหนดค่าเริ่มต้นให้กับวินโดว์จำนวน 4 ชุด ที่ต้องใช้กับฟังก์ชันเหล่านั้น และจะได้พูดถึงเรื่องของฟังก์ชัน 'make__tsrwindow ( )' ในช่วงต่อไปซึ่งระบบวินโดว์นี้จะทำให้เราสามารถสร้างวินโดว์ได้อย่างรวดเร็ว เมื่อมีการเรียกใช้ป๊อปอัพฟังก์ชันโดยวินโดว์ทั้งหมดจะถูกสร้างขึ้นในระหว่างการกำหนดค่าเริ่มต้น

ในขั้นตอนสุดท้ายที่ออกจากการทำงานของ TSR จะทำด้วยการเรียกใช้ฟังก์ชัน 'keep ( )' ของเทอร์โบซีที่จะทำการประมวลผลฟังก์ชันของดอสเพื่อจะฝังตัวอยู่ในหน่วยความจำต่อไป (DOS terminate-and-stay-resident function) ซึ่งเมื่อทำการประมวลผลฟังก์ชันนี้แล้วจะคืนการควบคุมให้กับดอส (แสดง command prompt)

รูปที่ 2 การกำหนดค่าตัวแปร (global variable) ให้กับโปรแกรมควบคุมแป้นพิมพ์

```
/* Set the vid_mem pointer to the start of video
   memory.
*/
void set_vid_mem(void)
{
  int vmode;

  vmode = video_mode();
  if((vmode!=2) && (vmode!=3) && (vmode!=7)) {
    printf("video must be in 80 column text mode");
    exit(1);
  }
  /* set proper address of video RAM */
  if(vmode==7) vid_mem = (char far *) MK_FP(0xB000, 0000);
  else vid_mem = (char far *) MK_FP(0xB800, 0000);
}


/* Returns the current video mode. */
video_mode(void)
{
  union REGS r;

  r.h.ah = 15;  /* get video mode */
  return int86(0x10, &r, &r) & 255;
}
```

ทำความรู้จักฟังก์ชัน

รู้จักกับฟังก์ชัน 'tsr__key__stroke ( )' การทำงานของฟังก์ชันชุดนี้คือการเรียกใช้คีย์บอร์ด ISR ชุดเดิม เพื่อให้ทำหน้าที่อ่านรหัส (key) จากแป้นพิมพ์และนำไปเก็บไว้ในหน่วยความจำ (keyboard buffer) ตามปกติ โดยไม่จำเป็นจะต้องให้ ISR ชุดใหม่ทำหน้าที่นี้ด้วยและเมื่อรหัสถูกนำเอาไปเก็บไว้ในหน่วยความจำแล้ว ฟังก์ชัน 'tsr-keystroke ( )' จะทำหน้าที่ต่อไป โดยจะตรวจสอบดูว่าใช่รหัสที่ใช้เรียก (hot key) โปรแกรม TSR หรือไม่

เทคนิคอย่างหนึ่งที่นำเอามาใช้กับฟังก์ชันคือ การอ่านข้อมูลจากหน่วยความจำ (buffer) โดยตรง โดยไม่เรียกใช้บริการจากไบออสหรือฟังก์ชันใด ๆ ของดอสซึ่งเราจะได้พูดถึงหลักการและเหตุผลของการใช้วิธีการแบบนี้ในช่วงต่อไป ส่วนชุดคำสั่งของฟังก์ชัน 'tsr__keystroke ( )' ได้แสดงไว้ในรูปที่ 3

ส่วนรับข้อมูลจากแป้น

หน่วยความจำส่วนที่ดอสใช้สำหรับข้อมูลจากแป้นพิมพ์ นั้นจะเป็นไปตามมาตรฐานของดอสก็จะสามารถรับตัวอักขระได้ครั้งละไม่เกิน 16 ตัว ทำให้สามารถพิมพ์ข้อความเข้าไปได้อย่างต่อเนื่อง รวมทั้งทำให้สามารถเรียกใช้คำสั่งต่าง ๆ ล่วงหน้าได้และเมื่อมีการกดแป้นพิมพ์จะมีการเรียกใช้อินเตอร์รัพต์ 9 เพื่อให้ ISR ที่ทำหน้าที่อ่านข้อมูลจากแป้นพิมพ์อ่านข้อมูลนั้นจากพอร์ตและนำไปเก็บไว้ในหน่วยความจำ

รูปที่ 3 โปรแกรมตรวจสอบรหัสแป้นพิมพ์เพื่อเรียกใช้ TSR

```
/* This is the function that intercepts the keystroke
   interrupt (int 9). */
void interrupt tsr_keystroke(void)
{
  int far *t2 = (int far *) 1050;  /* address of head pointer */
  char far *t = (char far *) 1050; /* address of head pointer */

  (*old_key)();  /* first, call old keystroke ISR */

  if(*t != *(t+2)) {/* if not empty */
    t += *t-30+5; /* advance to the character position */
    if(*t>=59 && *t <=63) {
      switch(*t) {  /* see what is in scan code */
        case 59:  /* F1 - help */
          key = 1;
          break;
        case 60:  /* F2 - file list */
          key = 2;
          break;
        case 61:  /* F3 - directory */
          key = 3;
          break;
        case 62:  /* F4 - dec to hex */
          key = 4;
          break;
        case 63: /* F5 - detache tsr */
          key = 5;
          break;
      }
      *(t2+1) = *t2;  /* zero buffer */
    }
  }
}
```

จากนั้นถ้ามีการเรียกใช้ฟังก์ชันดอสหรือไบออสให้อ่านข้อมูลจากแป้นพิมพ์ก็จะนำเอาข้อมูลที่อยู่ในหน่วยความจำมาใช้ เพราะฉะนั้นจึงทำให้เราสามารถที่จะเขียนชุดคำสั่งต่าง ๆ เพื่อทำหน้าที่เหล่านี้แทนดอสและไบออส ซึ่งจะทำให้ TSR สามารถตรวจสอบได้ว่าเป็นคำสั่งที่เรียกใช้หรือไม่

สำหรับหน่วยความจำส่วนนี้จะถูกจัดสรรให้ใช้ตำแหน่ง 0000 : 041E (1054 ในเลขฐาน 10) เพราะการกดแป้นพิมพ์แต่ละครั้งจะทำให้มีการสร้างรหัส (16-bit scan code) ขึ้นมา ซึ่งหมายความว่าจะต้องใช้เนื้อที่ทั้งหมด 30 ไบต์ เพื่อเก็บข้อมูลจำนวน 15 ตัวอักษร แต่ปกติแล้วจะจัดเตรียมเนื้อที่เอาไว้ 32 ไบต์ เพื่อเตรียมไว้ให้กับรหัสปัดแคร่ (carriage return key) ที่จะถูกเพิ่มเข้าไปท้ายบัฟเฟอร์โดยอัตโนมัติ

การใช้งานหน่วยความจำส่วนนี้จะมีลักษณะเป็นการวนรอบตามลำดับก่อนหลัง (circular queue) โดยมีการกำหนดตำแหน่งที่เป็นหัวและท้ายคิวเอาไว้ โดยส่วนหัวจะทำหน้าที่ชี้ไปที่อักขระที่ถูกพิมพ์เป็นตัวสุดท้าย ในขณะที่ส่วนท้ายจะชี้ไปที่อักขระตัวต่อไปที่จะส่งไปให้กับดอสหรือไบออส เมื่อมีการเรียกใช้สำหรับตัวชี้ที่ส่วนหัวจะถูกเก็บเอาไว้ในตำแหน่ง 0000 : 041A (1050 ในเลขฐาน 10) และตัวชี้ส่วนท้ายจะถูกเก็บเอาไว้ในตำแหน่ง 0000 : 041C (1052 ในเลขฐาน 10) ซึ่งค่าของตัวชี้ทั้ง 2 ส่วนนี้จะเป็นตำแหน่งของข้อมูลในคิว หมายความว่าค่าที่เก็บเอาไว้นั้นจะเป็นตัวชี้ตำแหน่งที่เป็นอยู่ขณะนั้น (current position) บวกด้วย 30 (เพราะวิธีการอ้างตำแหน่งของไมโครโปรเซสเซอร์ 8086 เป็นแบบ indirect addressing) และในกรณีที่หน่วยความ-

จำส่วนนี้ว่าง (ไม่มีข้อมูล) ค่าของตัวชี้ทั้งหัวและท้ายจะเป็นค่าเดียวกัน

ในส่วนของเหตุผลที่ให้ฟังก์ชัน 'tsr__keystroke' ( )' ทำหน้าที่ตรวจสอบข้อมูลในหน่วยความจำส่วนนี้โดยไม่เรียกใช้ด้วยวิธีการปกติ ซึ่งมีอยู่ด้วยกัน 2 ประการคือ

- ประการแรกเป็นเรื่องของความเร็วในการทำงานของระบบ เพราะการอ่านข้อมูลเองโดยตรงจะทำงานได้เร็วกว่าการเรียกใช้บริการจาก OS (Operating System) และถ้ามีโปรแกรม TSR อยู่หลายชุดทำงานอยู่ในหน่วยความจำก็จะยิ่งทำให้การทำงานช้ายิ่งขึ้น

- เหตุผลอีกอย่างหนึ่งคือ วิธีการนี้จะช่วยให้สามารถแก้ไขข้อมูลในหน่วยความจำได้ เพราะถึงจะไม่เป็นประโยชน์กับ TSR แต่ก็จะเป็นประโยชน์กับโปรแกรมแอพพลิเคชันอื่น ๆ

โปรแกรม TSR ของเราจะใช้ฟังก์ชันคีย์จำนวน 5 ตัวคือ F1-F5 เพื่อทำหน้าที่เรียกใช้ TSR แต่ถ้าต้องการจะเปลี่ยนไปใช้แป้นอื่นก็ทำได้ เช่นอาจใช้ ALT-F1 ถึง ALT-F9 แทนได้ เพื่อหลีกเลี่ยงการใช้ฟังก์ชันคีย์ที่มักจะถูกนำไปใช้งานในโปรแกรมต่าง ๆ อยู่เสมอ อย่างไรก็ตามในช่วงการทดสอบเพื่อทดลองใช้งานโปรแกรม TSR นั้น การเรียกใช้ด้วยการกดแป้นเพียงครั้งเดียวย่อมสะดวกกว่า

ส่วนวิธีการที่ใช้ในการเขียนโปรแกรมเรียกฟังก์ชันคีย์คือเปลี่ยนค่าสแกนโค้ด (scan code) ให้เป็นรหัสใหม่ เช่น F1 นั้นมีค่าสแกนโค้ดเป็น 59 และ F2 จะมีค่าสแกนโค้ดเป็น 60 และเรียงต่อกันไปเช่นนี้เรื่อย ๆ โดยเราจะเปลี่ยนค่าสแกนโค้ดเหล่านี้ให้เป็นอย่างอื่นด้วยการใช้ตัวแปรที่มีค่าเริ่มต้นจาก 1 แทนค่าสแกนโค้ดของ F1 เรื่อยไปตามลำดับ ซึ่งเหตุผลที่ต้องใช้ค่าตัวแปรแทนสแกนโค้ดคือจะทำให้สามารถเปลี่ยนแป้นเรียกใช้ TSR ได้สะดวกขึ้น เพราะสามารถทำได้ด้วยการแก้ไขโปรแกรมเพียงบรรทัดเดียว

เมื่อมีการกดแป้นที่ใช้เรียก TSR จะต้องมีวิธีการลบรหัสในหน่วยความจำหลังจากอ่านค่าแล้ว ซึ่งวิธีการที่เร็วที่สุดคือการกำหนดให้ค่าของส่วนหัวของคิวมีค่าเท่ากับส่วนท้ายเหมือนกับที่ทำในฟังก์ชัน 'tsr__keystroke ( )'

จุดสำคัญอย่างหนึ่งเกี่ยวกับฟังก์ชัน 'tsr__keystroke ( )' คือตัวของมันเองไม่ได้ทำหน้าที่เรียกใช้โปรแกรม TSR จริง ๆ เพราะโดยทั่วไปแล้ว จะไม่สามารถขัดจังหวะการทำงานของโปรแกรมแอพพลิเคชันบางชุดได้ในระหว่างที่มีการใช้งาน keystroke interrupt เช่นถ้ามีความพยายามที่จะเรียกใช้ TSR จาก keystroke interrupt ระหว่างที่กำลังใช้งานเทอร์โบซี Turbo C integrated environment อยู่ก็อาจจะทำให้การทำงานของระบบเกิดความเสียหายได้ นี้คือเหตุผลที่จะต้องใช้ไทม์เมอร์อินเตอร์รัพต์เพื่อเรียก TSR เมื่อยังไม่มีการเรียกใช้ DOS-idle interrupt

อินเตอร์รัพต์ชุดใหม่

อินเตอร์รัพต์ชุดใหม่ของอินเตอร์รัพต์ 8 ในช่วงก่อนเราได้พูดถึงการสร้างไทม์เมอร์ฟังก์ชันขึ้นมาใหม่เพื่อใช้แทนของเดิมซึ่งฟังก์ชัน 'new__int 8 ( )' ทำหน้าที่คือการเรียกใช้รูทีนชุดเก่าก่อน (ซึ่งจุดนี้มีความสำคัญมากเพราะถ้าการเรียกใช้ล้มเหลวก็จะทำให้ระบบการทำงานเสียหายได้) จากนั้นจะทำการตรวจสอบการทำงานของดอสว่าว่างอยู่หรือไม่ ซึ่งถ้าว่างก็จะลบค่าที่มีอยู่ในตัวแปร 'busy' และถ้าพบว่ามีการกดฮอตคีย์ (มีรหัสของฮอตคีย์อยู่ในตัวแปร 'key' จะเรียกโปรแกรม TSR ให้ด้วยการเรียกใช้ฟังก์ชัน 'activate__sctsr' อีกต่อหนึ่ง

สำหรับตัวแปร 'busy' จะถูกนำมาใช้เพื่อให้เกิดความแน่ใจว่าจะไม่มีการเรียกใช้ TSR ในขณะที่มันกำลังทำงานอยู่ (ถูกเรียกใช้อยู่ก่อนแล้ว) เพราะดอสจะไม่ยอมให้มีการเรียกใช้ซ้ำ นอกจากนี้ stack-switching scheme ก็ไม่ยอมให้เรียกใช้ซ้ำด้วย ซึ่งการเรียกใช้ TSR ในขณะที่มันกำลังทำงานอยู่แล้วนั้น จะทำให้ระบบการทำงานเสียหาย จึงจำเป็นจะต้องทำการป้องกันเหตุการณ์นี้ไม่ให้เกิดขึ้น โดยค่าของ 'busy' จะถูกกำหนดให้จากฟังก์ชัน 'activate__sctsr ( )' เมื่อมีการเรียกใช้ TSR และจะลบค่านั้นทิ้ง เมื่อเลิกใช้ TSR แล้ว ซึ่งวิธีการนี้จะทำให้สามารถป้องกันการเรียกใช้ TSR ในระหว่างที่มันกำลังทำงานอยู่ได้

ส่วนชุดคำสั่งของไทม์เมอร์อินเตอร์รัพต์ชุดใหม่คือ 'new__int8 ( )' ซึ่งจะแสดงวิธีการใช้งานตัวแปร 'busy' ด้วยนั้น จะเป็นไปดังนี้

```
/* busy is set to 1 when the program is
active, 0 otherwise */
char busy = 0 ;
/* New interrupt 8, */
void interrupt new__int 8 (void)
{
      (*old__int 8) ( ) ;
      if (:* dos__active && ; busy && key)
            activete__sctsr ( ) ;
}
```

dos__idle ( ) ISR

ลักษณะการทำงานของ 'dos__idle ( )' นี้ก็จะเหมือนกับ 'new__

int 8 ( )' โดยจะเริ่มด้วยการเรียกใช้อินเตอร์รัพต์ 0×28 ISR ชุดเดิมก่อนแล้วจึงตรวจดูตัวแปร 'busy' และถ้าพบว่าไม่มีการกำหนดค่าเอาไว้ (ไม่ได้อยู่ในระหว่างการทำงานของ TSR และมีการกดแป้นเพื่อเรียกใช้ TSR ก็จะสามารถเรียกใช้ TSR ได้ ซึ่งชุดคำสั่งของฟังก์ชัน 'dos_idle' ( )' ก็จะเป็นไปตามที่จะแสดงไว้ต่อไปนี้

```
/* This function intercepts int 0×28 (the
dos-idle interrupt)
*/
void interrupt dos_idle (void)
{
      (*old_int28) ( )
      /* call old int 0×28 ISR */
      if (; busy && key) activete_sctsr
      ( ) ;
}
```

ประตูใหญ่ของ TSR

จุดที่จะทำให้ TSR สามารถผ่านเข้าไปทำงานในระบบได้คือ การผ่านเข้าไปด้วยฟังก์ชัน 'activate_sctsr ( )' ซึ่งจะทำหน้าที่เรียกฟังก์ชันต่าง ๆ ที่เกี่ยวข้องกับการกดแป้นฮอตคีย์ แต่ก่อนที่จะเรียกใช้ฟังก์ชันที่ต้องการก็จะต้องทำงานที่มีความสำคัญมากอย่างหนึ่งก่อน ซึ่งคือการจัดหาสแต็กไว้ให้ TSR ใช้ 1 ชุด เพราะเมื่อมีการเรียกใช้อินเตอร์รัพต์นั้น ดอสจะเรียกใช้สแต็กของมันเอง ซึ่งมีขนาดเล็กมาก ถึงแม้ว่ามันจะสามารถเก็บ far pointer ของฟังก์ชัน 'tsr_keystroke( )' ได้ 2 ชุด กับข้อมูลบางอย่างที่อินเตอร์รัพต์สร้างขึ้นเอง แต่ก็ยังเล็กเกินไปที่จะนำมาใช้กับโปรแกรม TSR เพราะฉะนั้นฟังก์ชัน 'active_sctsr ( )' จึงต้องจัดหาสแต็กไว้ให้ใช้และคืนให้กับระบบเมื่อออกจาก TSR

สำหรับวิธีการแบบนี้จำเป็นต้องใช้ตัวแปรรีจิสเตอร์ของเทอร์โบซีด้วย ฉะนั้นถ้าคอมพายเลอร์ตัวที่จะนำมาใช้นั้นไม่มีขีดความสามารถแบบนี้อยู่ด้วย ก็จะต้องใช้ภาษาแอสแซมบลีเพื่อสร้างสแต็กเอาไว้ใช้แทน สำหรับสแต็กของ 'SCTSR' จะถูกกำหนดเอาไว้ใน unsigned character array ชื่อ 'stack' โดยจะมีขนาด 0×2000 ไบต์ ซึ่งชุดคำสั่งที่จะใช้เรียก TSR (SCTSR) นั้นได้แสดงเอาไว้ในรูปที่ 4

ในการเปลี่ยนไปใช้งานสแต็กนั้น จะทำได้ด้วยการเรียกใช้ฟังก์ชัน 'disable ( )' ซึ่งเป็นฟังก์ชันของเทอร์โบซีที่ใช้ยกเลิกอินเตอร์รัพต์เพื่อไม่ให้มีการอินเตอร์รัพต์เกิดขึ้นจนกว่าการจัดการกับสแต็กจะเรียบร้อย ซึ่งจะทำให้การทำงานของระบบเป็นไปอย่างถูกต้องและเมื่อจัดการกับสแต็กเรียบร้อยแล้วก็จะยอมให้อินเตอร์รัพต์ได้ด้วยการเรียกใช้ฟังก์ชัน 'enable ( )' ซึ่งถ้าคอมพายเลอร์ที่ใช้เป็นตัวอื่น (ไม่ใช่เทอร์โบซี) และไม่มีฟังก์ชันเหล่านี้ก็จำเป็นจะต้องใช้ภาษาแอสแซมบลีที่ทำหน้าที่แทน คือ 'CLI' และ 'STI'

ขั้นตอนการทำงานต่อไปของฟังก์ชัน 'activate_sctsr ( )' คือการตรวจสอบวิดีโอโหมดที่ใช้อยู่ เพื่อให้แน่ใจว่า TSR ไม่ได้ถูกเรียกใช้ในกราฟิกโหมด (เพราะโปรแกรม TSR ของเราทำงานในเท็กซ์โหมด) และถ้าพบว่าเป็นวิดีโอโหมดที่ถูกต้องและ TSR ยังไม่

รูปที่ 4 แสดงโปรแกรมเรียกใช้ TSR

```
/* Pop up the SCTSR. */
void activate_sctsr(void)
{
  /* setup TSR stack */
  disable();  /* disable interrupts */
    ss = _SS;
    sp = _SP;
    _SS = _DS;
    _SP = (unsigned) &stack[STK_SIZE-2];
  enable();   /* enable interrupts */

  /* check video mode - don't popup if not in
     an 80-column text mode
  */
  video = video_mode();
  if(!busy && (video==7 || video==3 || video==2)) {
    busy = !busy;  /* don't allow a second activation */
    switch(key) {
      case 1: do_help();
        break;
      case 2: do_filelist();
        break;
      case 3: dirlist();
        break;
      case 4: dectohex();
        break;
    }
    busy = !busy;
  }
```

รูปที่ 4 (ต่อ)

```
    if(key==5){ /* deactivating TSR */
      cursor_pos();  /* get current cursor position */
      write_string(old_col, old_row, "SC-TSR deactivated", NORM_VID);
    }

    /* restore old stack */
    disable();
      _SP = sp;
      _SS = ss;
    enable();

    if(key==5) {  /* reset old ISR routines and free memory */
      setvect(8, old_int8);
      setvect(9, old_key);
      setvect(0x28, old_int28);
      setvect(0x64, old_int64);
      freemem(_psp);  /* free program block */
    } else  key = 0;
}
```

เริ่มทำงาน มันก็จะเรียกใช้ฟังก์ชันที่เกี่ยวข้องกับการกดแป้นฮอตคีย์

วิธีการที่จะตรวจสอบว่ามีการกดแป้นฮอตคีย์หรือไม่ก็จะดูได้จากค่าของตัวแปร 'key' ซึ่งจะมีค่าเป็น '0' ถ้าไม่มีการกดแป้นฮอตคีย์ ถ้าเป็น (F5) ถูกกด ก็จะแสดงว่ามีการสั่งเลิกใช้ TSR แล้ว ซึ่งการเลิกใช้ TSR ก็มีขั้นตอนสำคัญอยู่ด้วย ซึ่งก็คือการคืนค่าเดิมให้กับระบบ โดยจะต้องทำขั้นตอนนั้นก่อนที่จะคืนอินเตอร์รัพต์รูทีนชุดเดิมให้กับระบบ อย่างไรก็ตามได้มีการแสดงข้อความให้รู้และเป็นเหตุผลที่ทำให้มีการเปรียบเทียบ 'key = = 5' ในโปรแกรมถึง 2 ครั้ง

ในส่วนของฟังก์ชัน 'cusor__pos ( )' จะทำหน้าที่กำหนดค่าให้กับตัวแปร ชื่อ 'old__col' และ 'old__row' ซึ่งก็คือตำแหน่งของเคอร์เซอร์ในขณะนั้นและค่านี้จะถูกนำมาใช้เพื่อแสดงข้อความบนจอภาพของฟังก์ชัน 'write__string' สำหรับฟังก์ชันนี้จะเป็นฟังก์ชันที่ทำงานในระดับต่ำที่จะมีอยู่ในระบบวินโดว์ของเราด้วย

ขั้นตอนสุดท้ายของฟังก์ชัน 'activate__sctsr ( )' คือการเปลี่ยนค่าของตัวแปร 'key' ให้เป็น '0' เพื่อให้รู้ว่าไม่มีการกดแป้นใด ๆ เพื่อเรียกใช้ TSR

เรื่องของวินโดว์

ก่อนที่จะพูดกันต่อไปเกี่ยวกับโปรแกรม TSR ของเรา (SCTSR) ก็ลองมาดูกันถึงเรื่องของระบบวินโดว์ที่จะนำมาใช้ใน TSR เพื่อทำความเข้าใจเกี่ยวกับวิธีการใช้และวิธีการเขียนโปรแกรมเพื่อเรียกใช้วินโดว์

ระบบวินโดว์ที่จะนำมาใช้นั้นจะอยู่บนพื้นฐานของตัวแปรประเภทโครงสร้างที่มีชนิดเป็น 'window__frame' ซึ่งจะมีลักษณะเป็นอะเรย์โดยแต่ละอะเรย์ของโครงสร้างนี้จะถูกเรียกว่า 'frame' ซึ่งจะทำหน้าที่เก็บข้อมูลของแต่ละวินโดว์ที่ใช้ในระบบและโครงสร้างของ 'window-frame' ก็จะเป็นไปตามที่แสดงไว้ดังนี้

```
struct window__frame {
        int startx, endx, starty endy ;
        /* corners of window */
        int curx, cury ;
        /* current position in window */
        unsingned char *p ;
        /* pointer to buffer */
        char *hesder ;
        /* header message */
        int border ;
        /* border on/off */
        int active ;
        /* on screen yes/no */
} frame [MAX__FRAME]
```

สำหรับค่าของ 'MAX__FRAME' จะต้องมีขนาดใหญ่พอที่จะเก็บวินโดว์ทั้งหมดที่ระบบกำหนดเอาไว้และในส่วนของ TSR ของเรา ค่านี้จะมีค่าเป็น 4 ซึ่งก่อนที่จะสามารถใช้งานวินโดว์ได้ก็จะต้องมีการกำหนดเอาไว้ก่อนแล้วโดยเรียกใช้ฟังก์ชัน 'make__maketsr-window ( )' ซึ่งจะมีโปรโตไทพ์ดังนี้

```
make__tsrwindow (
        int num,
        /* window mumber */
        char * header,
        /* header text */
        int startx, int starty,
        /* upper left corner */
        int endx, int endy,
        /* lower right corner */
        int border
        /* no border if 0 */
```

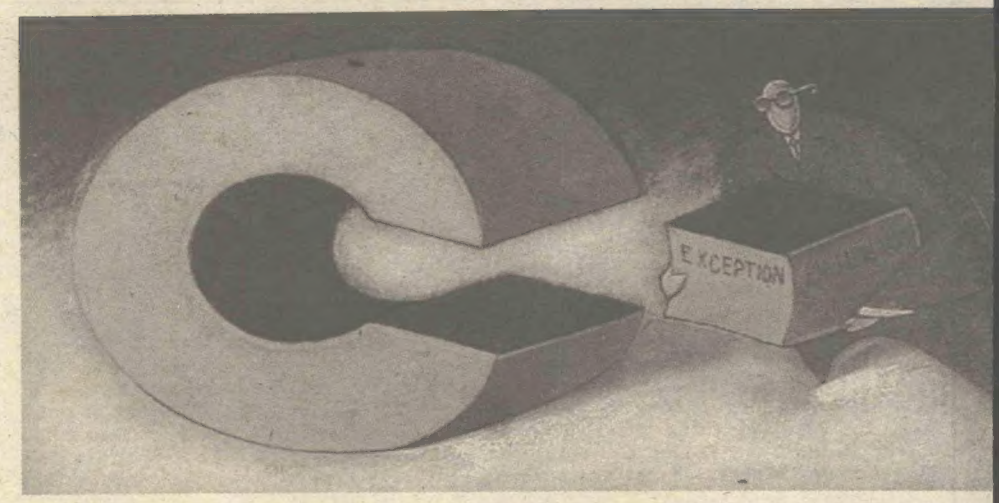
การเข้าถึงวินโดว์ทั้งหมดจะทำได้จากการระบุหมายเลขวินโดว์เอาไว้ในตัวแปร 'num' ส่วนขนาดของวินโดว์ก็จะกำหนดโดยใช้ขนาดของเส้นทแยงมุมของวินโดว์ซึ่งการเขียนข้อความต่าง ๆ นั้นจะทำได้เฉพาะภายในขอบเขตของวินโดว์เท่านั้น แต่ถ้าต้องการจะเขียนข้อความนอกวินโดว์ก็ทำในลักษณะของข้อความ (string) เอาไว้ใน 'header field' สำหรับขอบของวินโดว์จะต้องกำหนดค่าให้กับตัวแปร 'border' ซึ่งจะต้องไม่ใช่ค่า 0 (non-zero) และในระบบวินโดว์นี้ตำแหน่งมุมบนด้านซ้ายของวินโดว์จะต้องกำหนดให้เป็น 0,0

สิ่งที่จะต้องจำไว้อย่างหนึ่งคือฟังก์ชัน 'make_tsrwindow ( )' จะทำหน้าที่เพียงการกำหนดวินโดว์เท่านั้น ฉะนั้นก่อนที่จะส่งผลลัพธ์ไปที่วินโดว์จะต้องเรียกใช้ฟังก์ชัน 'tsrwindow ( )' พร้อมด้วยจำนวนวินโดว์ที่ต้องการจะเรียกใช้ ซึ่ง 'tsrwindow ( )' จะเก็บรายละเอียดบนจอภาพในส่วนที่วินโดว์จะนำมาใช้เอาไว้ในตัวแปรชื่อ 'wp' ให้โดยอัตโนมัติ อย่างไรก็ตามการใช้งานวินโดว์นั้นจะทำได้เพียงครั้งละ 1 วินโดว์เท่านั้น ซึ่งถ้าต้องการจะให้สามารถใช้งานวินโดว์ได้มากกว่าครั้งละ 1 วินโดว์จะต้องเตรียมหน่วยความจำเอาไว้หลายชุด

การยกเลิกวินโดว์และนำเอาจอภาพเดิมมาแสดงไว้แทนนั้นจะทำได้โดยใช้ฟังก์ชัน 'deactiveate ( )' ซึ่งจะต้องใช้จำนวนวินโดว์ที่ต้องการจะยกเลิกมาเป็นอาร์กิวเมนต์

ฟังก์ชันอื่น ๆ ของวินโดว์ที่เหลือจะทำหน้าที่จัดการเกี่ยวกับอุปกรณ์รับข้อมูล (console I/O) ตัวอย่างเช่นฟังก์ชัน 'tsrwindow_gets ( )' จะทำหน้าที่รับข้อความเพื่อนำไปแสดงในวินโดว์ที่กำหนดให้ ส่วนการกำหนดตำแหน่งของเคอร์เซอร์ในจุดที่ต้องการภายในวินโดว์จะทำได้จากฟังก์ชัน 'tsrwindow_xy ( )' ซึ่งตำแหน่งของเคอร์เซอร์นั้นจะมีความสัมพันธ์เฉพาะกับวินโดว์ที่ใช้เท่านั้น (ไม่ใช่จอภาพทั้งหมด)

สิ่งที่จะต้องจำไว้อีกอย่างหนึ่งคือทุก ๆ ฟังก์ชันที่มีชื่อเริ่มต้นเป็น 'tsrwindow' นั้น จะต้องมีการระบุจำนวนวินโดว์ที่กำลังใช้อยู่เอาไว้ให้เป็นอาร์กิวเมนต์ตัวแรกเสมอ



พักตรงนี้...ดีกว่า

ก็คงจะต้องหยุดกันไว้แค่นี้ก่อนแล้วคราวหน้าเราก็จะมาคุยกันต่อเป็นช่วงสุดท้ายของบทความชุดนี้ ก็ขอให้อดใจรออีกนิด อย่าเพิ่งเบื่อกันไปก่อนก็แล้วกัน เพราะคราวต่อไปจะเป็นเรื่องของการเขียนโปรแกรมยูทิลิตี้อีก 4 ชุด สำหรับ TSR ของเรา ซึ่งแต่ละชุดก็ล้วนแล้วแต่เยี่ยม ๆ ด้วยกันทั้งนั้น ไม่ว่าจะเป็นเรื่องหน้าที่ของโปรแกรม (ประโยชน์ใช้สอย) ขีดความสามารถ หรือโอกาสที่จะนำไปใช้ประโยชน์สำหรับงานส่วนตัวของคุณ และสิ่งที่สำคัญที่สุดคือวิธีการเขียนโปรแกรมเพื่อจะให้คอมพิวเตอร์ทำงานในลักษณะต่าง ๆ เหล่านั้น เพราะถ้าหากเข้าใจวิธีการต่าง ๆ ที่นำมาใช้เพื่อเขียนโปรแกรมแล้ว การที่จะนำไปประยุกต์ใช้เพื่อเขียนโปรแกรม TSR ให้ทำงานในลักษณะอื่นที่แตกต่างออกไปก็สามารถทำได้อย่างสะดวก ฉะนั้นคราวนี้ก็ลากันไปแค่นี้ก่อน แล้วค่อยพบกันใหม่นะครับ...สวัสดี